Homework 5: Q2

Name: Waiwai Kim, waiwaiki

1 Part (a): Proof Idea

The majority of proving that "flipping" each label still results in a consistent labeling is presented in Week 7 recitation notes. We are asked to prove that the case L' will label i and j differently.

(i, j) are labeled different. The fact that L is consistent implies the following:

- $L(i)$ = moth and $L(j)$ = butterfly. Flipping the labels results in $L'(i)$ = butterfly and $L(j)$ = moth. Thus, it is still true that $L'(i)$!= $L'(j)$. $L'(i)$ and $L'(j)$ are different.
- $L(i)$ = butterfly and $L(j)$ = moth. Flipping the labels results in $L'(i)$ = moth and $L(j)$ = butterfly. Thus, it is still true that $L'(i)$!= $L'(j)$. $L'(i)$ and $L'(j)$ are different.

Note that this is one of m definitive judgments. Because it's a definitive judgment, the two cases of different labels are exhaustive and we have shown that flipping a different label still results in a consistent label.

2 Part (b): Algorithm Idea

Given n specimens and m definitive judgments, we can construct an undirected graph $G = (V, E)$, in which vertex V represent specimens and an edge represents a definitive judgment. Different from a undirected graph we have seen in class so far, this graph will have two different types of edges, 1) $same$ and 2) $different$. We are agnostic to $ambiguous$ edges.

Next, we pick an arbitrary starting vertex called s on the graph G. If we assume that the graph is connected, picking an arbitrary starting vertex is sufficient. If the graph is not connected, we will pick an arbitrary starting vertex per each component.

We will run the BFS on G starting at s. We will also have a visit list of n nodes. When the BFS visits a vertex for the first time, we will label vertex either $same$ or $different$ depending on the edge the BFS has just traversed. For example, if the BFS visits a node v_i from the starting vertex s through a $same$ edge, we will denote the vertex as $same$ and vice versa. There may be some nodes that do not have any definitive judgment edge. These vertices are $ambiguos$.

Once the BFS is finished, we will enumerate each m definitive edges and examines if the BFS label are consistent with the edges. In the BFS, the algorithm uses a subset of edges in order to label. In this step, we determine if the BFS label from the previous step is consistent with every m edge. For example, it is inconsistent when $v_j = same$ and $v_k = different$ when $(v_j, v_k) = same$. If an inconsistent label exists, the algorithm returns FALSE. If no inconsistent label is found at the end of m enumeration, the algorithm returns TRUE.

3 Part (b): Algorithm Details

The pseudo code of the algorithm is presented in Algorithm 1.

- graph - input - adjacency list as a dictionary in Python.

- edge_list - input - edge list as a dictionary with a tuple as a key in Python. edge_list[(u,v)] will tell if the edge is $same$ or $different$.
- q - FIFO queue in order to implement BFS.
- visit_list = list of n length. It will tell if a vertex has been visited before.
- label_list - store label at a node u. Length of n.
- starting node - note that the algorithm arbitrary assigns moth to the starting node. This is not a problem as part(a) proves "flipping" doesn't change the consistency of a labeling.

4 Part (b): Proof of Correctness Idea

We will prove the algorithm is correct based on the material presented in 3.4 of the textbook. We can think of our graph as bipartite as well because we can think of $different$ edge as between level $L[i]$ and $L[i+1]$, while $same$ edge between two vertices in the same layer. Thus, we will use (3.15) presented in the textbook. Let G be a connected graph, and $L_1, L_2, ...$ be the layers produced by BFS starting at node s through $different$ edges. Then exactly one of the following two must be true.

- There is no $different$ edge between two nodes of the same layer. In this this G is a bipartite graph in which the even number layers are moth and the nodes in odd-numbered layers are butterfly.
- There is a $different$ edge of G joining two nodes of the same layer produced by the BFS. Is this G contain an odd-different-length cycle, and it cannot be bipartite. Note that it cannot be the case if $(v_i, v_j) = (v_j, v_k) = (v_k, v_i) = different$ because $v_i = moth \rightarrow v_j = butterfly \rightarrow v_k = moth \rightarrow v_i = butterly$. There is a contradiction at v_i.

5 Part (b): Proof Details

The proof details are presented in the textbook 3.4 page 96. First, in the first bullet point, we know that every edge of G joins nodes either in the same layer (same) or in adjacent layer (different). The first bullet point assumptions the opposite of these cases don't happen. This means that every different edge joins two nodes in adjacent layers. Our labeling procedure gives nodes in adjacent layers the opposite of the previous layer, and so every different edge has ends with opposite specimen. Thus, this labeling established that G is consistent.

Regarding the second bullet point where there is an odd cycle of different edges. We are told that G contains a different edge joining two nodes of the same layer. Refer to the Figure 3.6 of the textbook on page 96. Imagine the edges shown in the figure are different. The proof in the textbook shows that there is an odd-numbered-cycle that renders our graph inconsistent.

6 Part (b): Runtime Analysis

As the textbook claims, the algorithm presented here is basically the BFS with a labeling logic implemented on top of BFS. In labeling, whenever we traverse through a different edge, we flips the labeling. At the end of the procedure, we simply scan all th edges and determine whether there is any inconsistent edge, which takes $O(m)$. Thus, the total running time for the algorithm is $O(m+n)$, as it is for BFS.

Algorithm 1 Algorithm1

```
1:  procedure ALGORIGHM-1(Inputs: Adjacency list, edge dictionary )
2:      q = queue()
3:      pick an arbitrary starting node
4:      q.append(starting node s)
5:      visit_list =[0]*len(graph)
6:
7:      label_list = [-1]*len(graph)
8:      label_list[starting node] = moth
9:
10:     inconsistent = []
11:     while q do not empty:
12:         current = q.pop()
13:         for i in graph[current]: do
14:             if visit_list[i] = 0 then
15:                 visit_list[i] = 1
16:                 if edge_dict[(current, i)] = same  then
17:                     label_list[i] = label_list[current]
18:                 else
19:                     label_list[i] = not(label_list[current])
20:                 end if
21:             else
22:                 pass
23:             end if
24:         end for
25:     end while  BFS is done
26:
27:     for each edge (u, v) in edge_dict.items(): do
28:         if edge_dict[(u,v)] == same: then
29:             if label_list[u] != label_list[v]: then
30:                 return FALSE
31:             else
32:                 pass
33:             end if
34:         end if
35:         if edge_dict[(u,v)] == different: then
36:             if label_list[u] == label_list[v]: then
37:                 return FALSE
38:             else
39:                 pass
40:             end if
41:         end if
42:     end for
          Return TRUE
43: end procedure
```